NOTES

SUR LE

VLISP 10.2

Mai 1977

Resume :

Modifications et ameliorations apportees a l'interprete VLISP-10.
Description de l'assembleur (LAP) et du compilateur (COMPIL).
Ces notes ne sont qu'une mise a jour du
manuel de reference VLISP-10.
RT 17-76, Departement d'Informatique,
Universite de Paris 8, Mars 1976.

Jerome CHAILLOUX

T A B L E   D E S   M A T I E R E S

SECTION PAGE

1 ORGANISATION 2

1.1 Utilisation de l'interprete 2
1.2 Le decoupage de la memoire 2
1.3 Le Garbage-Collecting 4
1.4 Les changements de configuration 5

2 L'INTERPRETE 7

2.1 EVAL ET APPLY 7
2.2 Les MACROs 7
2.3 Les tableaux 9
2.4 Le Mode AUTOLOAD 12

3 LES NOMBRES 13

3.1 Les conversions 14
3.2 Arithmetique generale 14
3.3 Fonctions mathematiques 16

4 LES FONCTIONS STANDARDS 19

4.1 Les fonctions modifiees 19
4.2 Les nouvelles fonctions 20
4.3 Les fonctions testant les types 22
4.4 Les fonctions sur les P-listes 23
4.5 Les fonctions sur pile 23

5 LES ENTREES-SORTIES 25

5.1 La fonction READ 25
5.2 La fonction PRINT 26
5.3 Les speficications de fichier 26
5.4 Fonctions sur les fichiers 27
5.5 Les fichiers image-memoire 28

6 LE PRETTY-PRINT 29

6.1 Introduction 29
6.2 Les Fonctions du PRETTY-PRINT 29

7 LE LAP 31

7.1 Les registres 31
7.2 Acces aux objets LISPs 32
7.3 Test de type 32
7.4 Creation d'objets 33
7.5 Format externe d'une instruction 33
7.6 Les pseudos-instructions 34
7.7 Les macros-LAP 35
7.8 Appel des fonctions 36
7.9 Les Fonctions standards du LAP 37
7.10 Exemples d'utilisation du LAP 37

8 LE COMPILATEUR 42

8.1 Les macros du compilateur 42
8.2 Points d'entree speciaux 43
8.3 Les Fonctions standards du compilateur 45
8.4 Exemples d'utilisation du compilateur 46

L'interprete VLISP-10 est utilise depuis un an et demi. Il est disponible dans plusieurs sites (I.R.C.A.M. , C.H.U. Salpetriere ...). Des erreurs y ont ete corrigees et de nouvelles possibilites y ont ete incluses en particulier :
- les nombres flottants
- les macros
- les tableaux
- une configuration initiale.

De plus un logiciel de base puissant autant que varie a ete ecrit, dans lequel se trouvent :
- des editeurs
- des correcteurs
- des aides a la mise au point
- un assembleur LAP
- un compilateur.

Grace soit rendue aux nombreux utilisateurs de VLISP-10 qui ont contribue a l'elaboration du nouvel interprete VLISP-10.2 .

Que les dieux aient en leur sainte garde :
- Patrick Greussay, le pere des VLISPs,
- Harald Wertz, le createur de PHENARETE,
- Daniel Goossens, le roi du meta-evaluateur.
- Brian Harvey, pour qui le PDP10 n'a, depuis longtemps, plus de secrets.

Vous qui trouvez des erreurs, des inexactitudes, des points obscurs, ou vous qui voulez simplement parlez de choses et d'autres, ...
vous pouvez toujours contacter :

Jerome Chailloux
Universite de PARIS 8
Route de la Tourelle
75012 Paris.
tel: 374 12 50 poste 299.

Vous etes les bienvenus ...

SECTION 1

ORGANISATION

L'interprete se trouve maintenant dans le segment haut (HIGHSEG) de la memoire. Sa taille actuelle est de 5 k (environ). Il est devenu partageable et reentrant. Les vertus d'une telle organisation sont multiples : elle permet entre autre de n'avoir qu'une copie de l'interprete en memoire dans le cas de plusieurs utilisateurs et d'accelerer les swapping du systeme qui, dans la mesure du possible, ne touchera pas a l'interprete lui-meme.

1.1 Utilisation de l'interprete

L'interprete VLISP-10.2 se trouve dans la zone systeme. Pour l'utiliser il suffit maintenant d'emettre la commande :

.R VLISP

Avant de commencer le dialogue sur les fichiers standards d'entree-sortie, le fichier initial CONFIG.INI est charge. Ce fichier peut ne pas exister dans la partition de l'utilisateur, en ce cas les options standards decrites ci-dessous seront prises. Ce fichier peut contenir une reconfiguration des zones de l'interprete ou des definitions de fonctions prealables a un travail donne ou bien encore une definition de travaux a executer en "batch".

Une fois ce fichier charge (ou non), l'interprete fonctionne en mode conversationnel : il lit une S-expression sur le fichier standard d'entree, l'evalue et imprime le resultat de cette evaluation sur le fichier standard de sortie. Si le bit 0 du R.G. est positionne (ce qui est l'option par defaut), le temps qu'a dure l'evaluation est egalement imprime. La frequence de l'horloge du PDP10 etant de 50 Hz, les durees de 0 et de 20 milli-secondes ne sont pas significatives. Tout ceci peut bien evidemment etre modifie au moyen des fonctions TOPLEVEL et CONFIGURATION.

1.2 Le decoupage de la memoire

La memoire est divisee en zones fixes (dont les tailles peuvent etre changees a l'initialisation du systeme). Il y a 8 zones dans le segment bas (LOWSEG) et une zone dans le segment haut (HIGHSEG).

Zone 0 du segment bas :

cette zone contient les memoires de travail de l'interprete (buffers, zones de sauvetages diverses ...) ainsi que certaines fonctions flottantes. La taille de cette zone varie entre 1 et 2 k en fonction du nombre de fonctions flottantes chargees.

Zone 1 du segment bas :
c'est dans cette zone que sont stockes les atomes litteraux a raison de 6 mots par atomes. La taille standard de cette zone permet de contenir 500 atomes. Il existe environ 250 atomes predefinis dans le systeme (les constantes et les fonctions standards), chaque utilisateur peut donc disposer d'environ 250 atomes litteraux pour ses besoins propres.

Zone 2 du segment bas :
c'est dans cette zone que sont stockes les nombres. Il existe 3 categories de nombres. a) Les nombres entiers qui possedent une representation unique et sont stockes sur un seul mot (d'une maniere standard tous les nombres compris entre -64 et +127 sont de ce type). b) Les nombres entiers qui peuvent avoir plusieurs representations simultanees et qui sont stockes sur 2 mots. c) Les nombres flottants qui peuvent avoir egalement plusieurs representations simultanees et qui sont stockes sur 2 mots. La taille standard de cette zone permet de stocker jusqu'a 1000 nombres de n'importe quel type.

Zone 3 du segment bas :
Cette zone contient les pointeurs de chaines. Chacun de ces pointeurs utilise 1 mot memoire. La taille standard de cette zone permet de stocker jusqu'a 100 pointeurs de chaines.

Zone 4 du segment bas :
Cette zone contient les doublets de liste stockes a raison d'1 mot par doublet. La taille standard de cette zone permet de stocker 8000 doublets de liste.

Zone 5 du segment bas :
Cette zone contient la pile de l'interprete et a une taille standard de 1200 mots.

Zone 6 du segment bas :
c'est dans cette zone qu'est stockee la pile utilisateur et les tableaux. Un element de pile ou un element de tableau occupe 1 mot memoire. La taille standard de cette zone est de 500 mots.

Zone 7 du segment bas :
c'est dans cette zone qu'est charge le code resultant du LAP ou du compilateur. Sa taille standard est de 200 mots.

Zone du segment haut :
Cette zone, partageable par plusieurs utilisateurs, contient l'interprete lui-meme. Sa taille actuelle se situe entre 5 et 6 k et n'est bien evidement pas compressible.

1.3 Le Garbage-Collecting

Chacune de ces zones est allouee d'une maniere specifique. Lorsque l'une de ces zones vient a saturation, une machinerie connue sous le nom de "Garbage-Collecting" (G.C.) est invoque, pour recuperer la place perdue. Chaque zone possede un dispositif de recuperation qui lui est propre. Si cet essai s'avere infructueux (une zone restant saturee) un message d'erreur est imprime. Ce type d'erreur est fatal ; il y a donc retour immediat au top-level. Il faut recommencer le travail en augmentant la taille de la zone incriminee.
Si vous interrompez l'interprete (en tapant un ^C) durant un G.C., le message suivant apparait :
^ G.C. ^ type .CONT please.
vous devez emettre la commande .CONT pour terminer le G.C. puis reinterrompre l'interprete. Si vous relancez l'interprete (par la commande .REE) qui n'a pas termine un G.C., vous le placez dans un etat de confusion epouvantable.

Messages relatifs au remplissage d'une zone memoire.

```
** no room for atoms.        remplissage de la zone atome.
** no room for numbers.      remplissage de la zone nombre.
** no room for strings.      remplissage de la zone chaine.
** no room for lists.        remplissage de la zone liste.
** no room for arrays.       remplissage de la zone tableau.
** no room for code.         remplissage de la zone code.
```

Il est possible d'avoir des statistiques a chaque "Garbage-Collecting" en positionnant le bit 5 du R.G. (plus precisement en evaluant la forme (STATUS 1 5) .). Apres chaque "Garbage-Collecting", le texte suivant sera imprime :

```
**** G.C. No             : xx
     G.C. Count          : xx
     ALTERED CELLS       : xx
     Lists          size : xxxx  marked : xxxx  freed : xxxx
     Strings        size : xxxx  marked : xxxx  freed : xxxx
     Numbers        size : xxxx  marked : xxxx  freed : xxxx
     Atoms          size : xxxx  marked : xxxx  freed : xxxx
     Code           size : xxxx  marked : xxxx  freed : xxxx
     Elapsed time        : xxxx  milli-sec.
     Average time in G.C. : xx %
```

Ces statistiques permettent de connaitre :
- le numero du garbage-collecting (le nombre de G.C. effectues depuis le lancement de VLISP).
- le compte actuel de G.C. (utilise par la fonction (STATUS 23))
- le nombre de doublets de listes irrecuperables. Si ce nombre n'est pas egal a 0, les G.C. suivant ne seront pas garantis ... Il vaut mieux recharger un nouvel interprete ...
- le nombre total d'atomes, utilises et libres.
- le nombre total de nombres, utilises et libres.
- le nombre total de chaines, utilisees et libres.

- l'occupation de la zone code
- le temps qu'a dure ce G.C.
- le pourcentage de temps passe uniquement dans le G.C. par rapport au temps d'utilisation total de l'interprete.

Dans la version I.R.C.A.M. de VLISP-10, un resume de ces statistiques comprenant : le nombre de G.C., le nombre de doublets libres, le nombre d'atomes libres et le pourcentage de temps utilise par le G.C., est systematiquement affiche sur la 3eme "who line" des ecrans fonctionnant en DM mode.

1.4 Les changements de configuration

Il est possible, en debut de travail, de changer la taille des zones standards. La fonction CONFIGURATION execute ce travail. Cette fonction, si elle est utilisee, doit etre la -premiere- et -derniere- forme a evaluer sur le fichier CONFIG.INI.

(CONFIGURATION init in out atoms nbs strgs lists stack array code)
[SUBR a 10 arguments]

cette fonction possede 10 arguments qui sont dans l'ordre :
- init : specification du nouveau fichier initial
- in : specification du fichier standard d'entree.
- out : specification du fichier standard de sortie.
- atoms : nombre total d'atomes.
- nbs : nombre total de nombres.
- strgs : nombre total de chaines.
- lists : nombre total de doublets.
- stack : taille de la pile de l'interprete.
- array : taille de la zone des tableaux.
- code : taille de la zone de stockage du code.

Ces arguments doivent etre donnes -dans l'ordre- . Le 1er argument est obligatoire et ne doit pas etre '(DSK (CONFIG . INI)) sous peine de faire boucler l'interprete durant le prelude. Si on ne desire pas modifier la taille d'une zone, l'argument a fournir est NIL. S'il n'est pas possible d'allouer la memoire demandee, le message suivant est imprime :

** not enough core.

et la configuration n'est pas modifiee.

exemples d'utilisation de la fonction de configuration.

```
; definition de la configuration standard ;
; cette configuration occupe :
     - 17 k en LOWSEG
     -  5 k en HIGSEG
 soit 22 k au total   ;
```

```
(CONFIGURATION
        '(DSK (VLISP . INI))      ; fichier initial ;
        '(TTY)                    ; fichier d'entree ;
        '(TTY)                    ; fichier de sortie ;
        500                       ; atomes litteraux ;
        1000                      ; nombres;
        100                       ;  chaines ;
        8000                      ; listes ;
        1200                      ; pile systeme ;
        500                       ; tableaux ;
        200                       ; code ;
)))))

; definition d'une petite configuration batch ;
; cette configuration occupe :
     - 10 k en LOWSEG
     -  5 k en HIGSEG
soit   15 k au total    ;

(CONFIGURATION
        '(DSK (VLISP . INI))      ; fichier initial ;
        '(DSK (LISP  . VLI))      ; fichier disque d'entree ;
        '(DSK (LISP  . LST))      ; fichier disque de sortie ;
        400                       ; 150 atomes utilisateur ;
        500                       ; 500 nombres ;
        50                        ; 50 chaines ;
        4000                      ; 4000 doublets de liste ;
        1000                      ; pile de l'interprete ;
        0                         ; pas de tableaux ;
        0                         ; pas de code ;
)))
```

SECTION 2

L'INTERPRETE

En plus d'ameliorations notables, l'interprete a subit un certain nombre de modifications. Ces modifications le rendent toutefois absolument compatible avec VLISP-10 version 1.

2.1 EVAL ET APPLY

Les fonctions de l'interprete EVAL et APPLY ont subi de nombreuses ameliorations qui les rendent a la fois plus puissantes et plus rapides.
Un mode de lancement rapide pour les fonctions de type SUBR ou FSUBR est utilise, si ces fonctions n'ont jamais ete redefinies.
Il est possible d'utiliser des nombres en tant que fonction tant pour EVAL que pour APPLY. La valeur retournee par ce type de fonction est le nieme element de la liste premier argument; le nombre n etant le nombre qui tient lieu de fonction. L'interprete assume donc un appel implicite de la fonction CNTH pour evaluer ce type de fonction.

```
ex : (3 '(A B C D E))  ->  C
     ((ADD1 3) '(A B C D E))  ->  D
     (APPLY 2 '((A B C D E)))  -> B
```

L'interpretation des fonctions tail-recursives est effectue d'une maniere tres efficiente sans consommation de pile [Greussay 1976].

```
ex : (DE FOO (X) (FOO X))   (FOO 1)
```

boucle a l'execution mais sans faire deborder la pile.
De meme que cet appel de bien mauvais aspect :

```
((LAMBDA (X) (X X)) '(LAMBDA (X) (X X)))
```

2.2 Les MACROs

L'interprete reconnait un nouveau type de fonctions, les MACROs. Une macro est un atome qui possede sur sa P-liste une lambda-expression sous l'indicateur MACRO. A l'evaluation d'une forme dont le CAR est une macro, EVAL evaluera d'abord la fonction associee a cette macro avec toute la forme comme argument, puis re-evaluera la valeur retournee de cette premiere evaluation. L'evaluation d'une macro se fait donc en deux temps.
C'est l'appel de la macro tout entier qui est passe en argument, il est donc possible de modifier physiquement cet appel a la premiere

evaluation de la macro (voir les exemples). Il existe une nouvelle fonction qui permet de definir aisement des MACROs a la maniere des fonctions DE DF DG DMI ou DMO.
Ces MACROs ne doivent pas etre confondues avec les macros-fonctions d'entree et de sortie (MACIN MACOUT) qui ne sont activees que pendant des lectures ou des ecritures.

(DM nom list-args s1 ... sN) [FSUBR]

definit une fonction de type macro. Cette fonction a pour nom nom, sa liste d'arguments est list-args et son corps de fonction est s1 ... sN. DM ramene le nom de la fonction en valeur, et peut etre definie comme suit :

```
(DF DM (L) (PUT (CAR L) (CONS 'LAMBDA (CDR L)) 'MACRO))
```

exemples d'utilisation des MACROs :

```
 ; liaison des arguments d'une MACRO ;

 ? (DM MAC (L)
 ?     (PRINT 'L '= L)      ; cette MACRO edite son argument et  ;
 ?     '(ADD1 3))           ; ramene la forme (ADD1 3) qui sera  ;
 = MAC                      ; reevaluee. ;

 ? (MAC 'X)
   L = (MAC 'X)
 = 4

 ; definition de la fonction MCONS sous forme de MACRO ;

 ? (DM MCONS (L)
 ?   (IF (NULL (CDDR L))
 ?       (CADR L)
 ?       ['CONS (CADR L) (CONS 'MCONS (CDDR L))])))
 = MCONS

 ? (MCONS 'A 'B 'C)
 = (A B . C)

 ; exemple de modification physique du corps des
   fonctions contenant des MACROs ;

 ? (DM ZEROP (L)
 ?   (RPLACA L 'EQ)
 ?   (RPLACD L [(CADR L) 0]))
 = ZEROP

 ? (DE FACT (N)
 ?   (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))
 = FACT

 ? (FACT 5)
 = 72
```

```
? (CDR 'FACT)
  (EXPR (LAMBDA (N)
    (IF (EQ N 0) 1 (TIMES N (FACT (SUB1 N))))))
```

2.3 Les tableaux

Un nouveau type d'objet est apparu, le tableau. Un tableau est un ensemble de memoires contigues pouvant contenir n'importe quel objet lisp (atomes, nombres, chaines ou listes). Les differents elements d'un tableau sont numerotes a l'aide d'un indice. Tous les tableaux sont actuellement uni-dimensionnes par souci d'efficacite, le premier indice est 0 (ces tableaux sont donc des vecteurs).
Les differents tableaux sont stockes dans une zone speciale; cette zone est partagee avec la pile utilisateur. Les fonctions sur les tableaux permettent de definir un tableau, d'avoir acces aux elements d'un tableau, de modifier un element d'un tableau ou d'appliquer une fonction aux differents elements d'un tableau.

(DA nom taille fonction) [SUBR a 3 arguments]

permet de definir un nouveau tableau. nom est un atome litteral qui devient le nom du tableau; taille est le nombre d'elements que peut contenir le tableau. S'il n'y a pas assez de place pour stocker ce nouveau tableau, l'erreur suivante apparait :
** no room for arrays.
Si la fonction (d'initialisation) n'est pas fournie, tous les elements du tableau sont initialises a NIL; dans le cas contraire, la fonction est appliquee pour chaque element avec l'indice de cet element lie au 1er argument (si il existe) de la fonction. Il peut donc y avoir un appel implicite de la fonction MAPARRAY. Les differentes valeurs ramenees par cette fonction seront les valeurs d'initialisation des elements du tableau. DA ramene le nom du tableau en valeur.

```
ex : (DA 'TAB 100)        ; definition d'un tableau de nom TAB de
                            100 elements. Chaque element est
                            initialise a NIL ;
     (DA 'TBL 10 '(LAMBDA (X) 0)) ; definition d'un tableau de
                                    10 elements, chaque element
                                    est initialise a 0 ;

     (DA 'ARR 10                 ; definition d'un tableau de 10
        '(LAMBDA (X) (- 9 X)))     elements initialises a :
                                   9 8 7 6 5 4 3 2 1 0 ;
```

Une fois le tableau defini, on a acces a chacun des elements du tableau en evaluant une forme a 1 argument dont la fonction est le nom du tableau et l'argument l'indice de l'element desire. Le nom du tableau devient donc une fonction d'acces a ce tableau. L'argument de cette fonction d'acces est evalue.

L'interprete ne teste la validite de l'indice fourni que si le bit 7 du R.G. est positionne (ce qui est l'option par defaut). Pour toutes les fonctions sur les tableaux qui vont etre decrites, les erreurs (le nom fourni n'est pas un nom de tableau ou l'indice est incorrect) sont signalees par le message :
 ** array error : <nom du tableau> <indice>
il est facile de definir de nouvelles fonctions d'acces a des tableaux deja definis ce permet de tester la validite du ou des indices (voir les derniers exemples).

```
ex : (TAB 5)  ->  NIL
     (TBL (ADD1 4))  -> 0
```

(DIM nom) [SUBR a 1 argument]

l'argument doit etre un nom de tableau. DIM ramene le plus grand indice de ce tableau.

```
ex : (DA 'TAB 50)  -> TAB
     (DIM 'TAB)    -> 49
```

(SETA nom indice valeur) [SUBR a 3 arguments]

met dans l'element du tableau specifie par le nom et l'indice, la valeur fournie en troisieme argument. SETA ramene en valeur la valeur chargee.

```
ex : (SETA 'TAB 5 10)  ->  10
     (TAB 5)  ->  10
```

(SETQA nom indice valeur) [FSUBR]

est identique a la fonction SETA mais le nom du tableau n'est pas evalue.

```
ex : (SETQA TAB 6 (ADD1 19))  -> 20
     (TAB 20)  ->  20
```

(MAPARRAY nom fonction) [SUBR a 2 arguments]

permet d'appliquer la fonction specifiee pour chacun des elements du tableau dont le nom est donne. Le 1er argument de la fonction est lie (s'il existe) a l'indice de l'element du tableau. MAPARRAY ramene NIL en valeur.

```
ex : (MAPARRAY 'TAB '(LAMBDA (X) (SETA 'TAB X (ADD1 X))))
       ->  NIL
```

(MAPARRAYQ nom fonction) [FSUBR]

est identique a la fonction MAPARRAY mais le nom du tableau n'est pas evalue.

(FILLARRAY nom liste) [SUBR a 2 arguments]

range dans les differents elements du tableau, dont le nom est specifie comme 1er argument, les elements successifs de la liste 2eme argument. Si la liste est trop courte, le reste du tableau est rempli avec des NILs ; si la liste 2eme argument est en realite un atome, le tableau est rempli avec cet atome. FILLARRAY ramene le nom du tableau en valeur.

```
ex : (DA 'TAB 10)  ->  TAB
     (FILLARRAY 'TAB '(0 1 2 3 4 5 6))  ->  TAB
     (FILLARAY 'TAB '99)  ->  TAB
```

(LISTARRAY nom) [SUBR a 1 argument]

ramene la liste constituee de tous les elements du tableau dont le nom est fourni en 1er argument.

```
ex : (DA 'TAB 5)   -> TAB
     (FILLARRAY 'TAB '(10 11 12))  -> TAB
     (LISTARRAY 'TAB)  -> (10 11 12 NIL NIL)
```

exemples d'utilisation des tableaux :

```
? (DA 'TAB 100)          ; definition d'un tableau de nom TAB  ;
= TAB                    ; possedant 100 elements, chacun des  ;
                         ; elements est initialise a NIL.      ;
? (DIM 'TAB)
= 99

? (DE TAB1 (I)           ; definition d'une nouvelle fonction  ;
?   (IF (LE 0 I 99)      ; d'acces a TAB1 qui teste la validite ;
?       (TAB I)          ; de l'indice fourni ;
?       (ERROR ["debordement TAB1 " I])))
= TAB

? (DE TAB2 (I J)          ; definition d'une nouvelle fonction ;
?   (IF (AND (LE 0 I 9)   ; permettant d'acceder au tableau    ;
?            (LE 0 J 9))  ; TAB au moyen de 2 indices pouvant ;
?       (TAB (PLUS (TIMES I 10) J))  ; varier de 0 a 9. Cette ;
?       (ERROR ["debordement TAB2 " I J]))  ; fonction teste ;
                                     ; les indices fournis ;
= TAB2
```

```
? (SETQA TAB 20 3)
= 3

? (TAB1 20)
= 3

? (TAB2 2 0)
= 3


? (DA 'CLAIR 10 '(LAMBDA (X)
?       ((ADD1 X) '("zero" "un" "deux" "trois" "quatre"
?                   "cinq" "six" "sept" "huit" "neuf"))))
= CLAIR

? (CLAIR 1)
= "un"

? (CLAIR 2)
= "deux"

? (MAPARRAYQ CLAIR (LAMBDA (X)
?      (SETQA CLAIR X (CONCAT "-" (CLAIR X) "-"))))
= NIL

? (CLAIR 9)
= "-neuf-"
```

2.4 Le Mode AUTOLOAD

Certaines fonctions utilisees frequemment, comme les fonctions d'editions ou de corrections, peuvent etre chargees automatiquement a leurs premier appel. Ces fonctions doivent se trouver dans un fichier disque et peuvent etre de n'importe quel type (EXPR FEXPR MACRO ARRAY SUBR compilee FSUBR compilee ...). Ce fichier sera lu par la fonction LIBRARY. Si la fonction ne se trouve pas dans le fichier indique, l'erreur ** undefined function apparait. La fonction AUTOLOAD permet de declarer des fonctions de ce type.

(AUTOLOAD file at1 at2 ... atN) [FSUBR]

met l'indicateur AUTOLOAD sur les differents atomes at1 ... atN. Ces fonctions devront se trouver dans le fichier de nom file. Au 1er appel de l'une de ces fonctions, le fichier file sera charge en entier.

ex : (AUTOLOAD PHENAR PHENARETE PHENARETEFILE) declare les deux fonctions PHENARETE et PHENARETEFILE de type AUTOLOAD, elles doivent se trouver dans le fichier PHENAR.

SECTION 3

LES NOMBRES

Il existe deux types de nombres : les nombres entiers et les nombres flottants. Ils sont stockess dans une zone de longeur fixe geree dynamiquement. Si cette zone se revele trop petite, le message d'erreur suivant apparait :
** no room for numbers.

La representation externe des nombres entiers est une suite de digits dans la base de numeration est definie par les fonctions (STATUS 5 n) en entree et (STATUS 6 n) en sortie ; par defaut, la valeur de ces bases est de 10. Ils peuvent etre signes ou non. Ces nombres sont stockes en memoire sur un mot (de 36 bits) ; ils doivent donc etre compris dans l'intervalle [-2 ^ 35 , +2 ^ 35 - 1] c'est-a-dire [-34359738368 , +34359738367].

La representation externe des nombres flottants est une suite de digits decimaux suivie immediatement par un point decimal (.) et suivie optionnellement par une fraction decimale egalement. Pour garder une precision de 8 digits un exposant peut etre rajoute en notation "E". Cette derniere forme d'ecriture n'est toutefois disponible qu'en sortie actuellement. Les nombres flottants sont stockes en memoire sur un mot (de 36 bits) qui contient 1 bit de signe, 8 bits d'exposant et 27 bits de mantisse.

Il n'existe a l'heure actuelle ni de nombres flottants "double precision", ni de nombres complexes.

Toutes les fonctions et predicats de VLISP-10.1, utilisant des nombres, sont disponibles et leurs utilisation deumeure inchangee.

Toutes les nouvelles fonctions qui vont etre decrites testent si leur(s) argument(s) sont des nombres. Dans le cas ou ils ne le seraient pas, une erreur apparait. Le libelle de cette erreur est :
<fonction> : ** non numeric arg : <argument>
dans lequel le nom de la fonction et l'argument incrimine sont imprimes.

Un certain nombre de fonctions mathematiques ont ete "empruntees" a la bibliotheque FORTRAN (ex: les fonctions SQRT, SIN, ATAN ...). Ces fonctions emetttent parfois des diagnostics d'erreurs qui leurs sont propres (par ex: la tentative de calcul de la racine carre d'un nombre negatif). Ces erreurs ne sont pas fatales sous VLISP mais le resultat du calcul est bien evidemment eronne.

3.1 Les conversions

(FIX n) [SUBR a 1 argument]

convertit le nombre flottant n en son equivalent entier. Un calcul d'arrondi est effectue automatiquement de la maniere suivante : la partie entiere est incrementee de 1 si la partie fractionnaire est >= 0.5 (pour les nombres negatifs le test est <= a 0.5).

```
EX :  (FIX 3.14)  ->  3
      (FIX 3.98)  ->  4
      (FIX 3)     ->  3
```

(FLOAT n) [SUBR a 1 argument]

convertit le nombre entier n en son equivalent flottant.

```
ex :  (FLOAT 3)            ->  3.
      (FLOAT -2)           ->  -2.
      (FLOAT 120000000)    ->  1.2E8
      (FLOAT 3.14)         ->  3.14
```

3.2 Arithmetique generale

Les arguments des fonctions qui vont etre decrites peuvent etre de n'importe quel type : entier ou flottant. Si l'un des arguments est flottant, le resultat sera un nombre flottant ; si tous les arguments sont de type entier, le resultat sera un nombre entier. Il y a donc conversion automatique des arguments avec priorite aux nombres flottants.

(1+ n) [SUBR a 1 argument]

ramene la valeur : n + 1 .

```
ex : (1+ 3)     ->  4
     (1+ 1.2)   ->  2.2
```

(1- n) [SUBR a 1 argument]

ramene la valeur : n - 1 .

```
ex : (1- 3)    ->  2
     (1- 3.2)  ->  2.2
```

(+ n1 n2) [SUBR a 2 arguments]

ramene la somme des 2 arguments : n1 + n2 .

```
ex : (+ 2 3)       ->  5
     (+ 2 3.2)     ->  5.2
     (+ 2.3 3)     ->  5.3
     (+ 2.3 3.3)   ->  5.6
```

(- n1 n2) [SUBR a 2 arguments]

ramene la difference des 2 arguments : n1 - n2 .

```
ex : (- 3 1)       ->  2
     (- 3 1.)      ->  2.
     (- 3.2 1.1)   ->  2.1
```

(* n1 n2) [SUBR a 2 arguments]

ramene le produit des 2 arguments : n1 * n2 .

```
ex : (* 3 2)       ->  6
     (* 3 2.1)     ->  6.3
     (* 3. 2.2)    ->  6.6
```

(// n1 n2) [SUBR a 2 arguments]

ramene le quotient des 2 arguments : n1 // n2 . Le caractere / est un caractere special, il faut le doubler pour utiliser cette fonction : on doit ecrire (// 3 2.) et non (/ 3 2.).

```
ex : (// 3 2)      ->  1
     (// 3 2.)     ->  1.5
     (// 1 3.)     ->  0.3333333
```

(** n1 n2) [SUBR a 2 arguments]

ramene la valeur de l'evaluation de n1 a la puissance n2.

```
ex : (** 2 3)      ->  8
     (** 2. 4)     ->  16.
     (** 2  5.)    ->  32.
     (** 2  0.5)   ->  1.414214
     (** 2 -0.5)   ->  0.7071068
     (** 15. 0)    ->  1.
```

(/\ n1 n2) [SUBR a 2 arguments]

ramene le reste de la division de n1 et n2. Cette fonction est utilisee en general avec des arguments entiers pour ramener la valeur du reste d'une division entiere (la fonction MODULO). Le caractere \ etant un macro-caractere, il faut le faire preceder du caractere / (quote caractere) pour utiliser cette fonction. On doit ecrire (/\ 5 3) et non (\ 5 3) .

```
ex : (/\ 5 3)     ->  2
     (/\ 5. 3.)   ->  2.980232E-8
```

3.3 Fonctions mathematiques

Pour toutes les fonctions qui vont etre decrites, l'argument s'il existe doit etre de type flottant. Il n'y a pas de conversion automatique.

(SQRT n) [SUBR a 1 argument]

ramene la racine carree du nombre n. n doit etre un nombre flottant positif.

```
ex : (SQRT 25.)   ->  5.
     (SQRT 5.)    ->  2.236068
     (SQRT -5.)   ->
%FRSLIB attempt to take SQRT of negative arg= 2.236068
```

(SIN n) [SUBR a 1 argument]

ramene la valeur du sinus de l'angle n exprime en radians.

(COS n) [SUBR a 1 argument]

ramene la valeur du cosinus de l'angle n exprime en radians.

(ATAN n) [SUBR a 1 argument]

ramene la valeur de la fonction arc tangente, c'est-a-dire la valeur de l'angle (en radians) dont la tangente est n.

```
; exemples d'utilisation des fonction trigonometriques ;

(SETQ PI   3.14159)     ->  3.14159
(SETQ PI:2 (// PI 2))   ->  1.570795
(SETQ PI:4 (// PI 4))   ->  0.7853975
```

```
(SIN 0.)                -> 0
(SIN PI:4)              -> 0.7071063
(SIN PI:2)              -> 1.

(COS 0.)                -> 1.
(COS PI:4)              -> 0.7071072
(COS PI:2)              -> 1.334181E-6

(SETQ X 1.2)            -> 1.2

(SQRT (+ (** (SIN X) 2) (** (COS X) 2)))  -> 1.

; la fonction TANGENTE n'est pas standard : definissons-la ;

(DE TANG (X)
   (// (SIN X) (COS X)))  -> TANG

(TANG 0.)               -> 0
(TANG PI:4)             -> 0.9999987
(TANG PI:2)             -> 749523.3

(ATAN 0.)               -> 0
(ATAN 1.)               -> 0.7853982
(ATAN (TANG 1.123))     -> 1.123
```

(EXP n) [SUBR a 1 argument]

ramene la valeur e puissance n.

```
ex : (EXP 0)      -> 1.
     (EXP 1.)     -> 2.718282
```

(LOG n) [SUBR a 1 argument]

ramene la valeur du logarithme neperien de n.

```
ex : (LOG 1.)          -> 0
     (LOG 2.718282)    -> 1.
```

(LOG10 n) [SUBR a 1 argument]

ramene la valeur du logarithme decimal de n.

```
ex : (LOG10 1.)        -> 0
     (LOG10 10.)       -> 1.
     (LOG10 20.)       -> 1.30103
     (LOG10 100.)      -> 2.
```

(RANDOM) [SUBR a 0 argument]

ramene a chaque appel un nombre flottant, aleatoire, compris dans l'intervalle]0. , 1.[.

```
ex : (RANDOM)  ->  0.1948187
     (RANDOM)  ->  0.7324636
     (RANDOM)  ->  0.6087399
```

SECTION 4

LES FONCTIONS STANDARDS

Quelques nouvelles fonctions sont apparues, et d'autres (tres peu nombreuses) ont ete modifiees, ce qui ne gene en aucun cas la compatibilite. Certaines fonctions d'entree-sortie ont egalement ete modifiees; une section leurs est consacree.

4.1 Les fonctions modifiees

(REVERSE l1 l2) [SUBR a 2 arguments]

La fonction REVERSE peut avoir maintenant un 2eme argument; s'il est fourni, cette fonction correspond a l'ancien appel :

(NCONC (REVERSE l1) l2).

S'il n'y a pas de 2eme argument son action est identique a l'ancienne version.

```
ex : (REVERSE '(A (B C) D E . F) '(G H I))
          -> (E D (B C) A G H I)
```

(LAST l n) [SUBR a 2 arguments]

De meme, pour la fonction LAST, un 2eme argument numerique peut lui etre ajoute pour obtenir la liste des n derniers elements de la liste l. Si n n'est pas donne, l'effet est inchange, et une liste contenant le dernier element de la liste argument est retournee comme valeur de la fonction. Cet ajout est tres utile, et permet notammnent de detruire physiquement le (ou les) dernier element d'un liste de la maniere suivante :

```
ex : (RPLACD (LAST '(A B C D) 2)) -> (A B C)
```

(OBLIST) [SUBR a 0 argument]

Il n'y a plus d'argument a cette fonction du fait de la gestion dynamique de la liste des objets internes. De plus OBLIST ne ramene plus l'atome UNDEF dans la liste des objets (ce qui evite bien des erreurs ** UNDEFINED VARIABLE).

4.2 Les nouvelles fonctions

(BOUNDP atome) [SUBR a 1 argument]

est un predicat qui teste si l'atome donne en argument possede une valeur (plus precisement possede une C-valeur differente de UNDEF). Ce predicat est utilise pour tester des variables libres sans crainte de l'erreur ** UNDEFINED VARIABLE.

ex : tester si l'atome X est lie, sinon lui donner la valeur NIL: (OR (BOUNDP 'X) (SETQ X NIL))

(COMMENT e1 e2 ... eN) [FSUBR]

est une fonction de commentaire qui ramene toujours l'atome COMMENT en valeur.

(COPY l) [SUBR a 1 argument]

fabrique une copie de la liste l en entier. Cette fonction est equivalente a l'ancienne ecriture (SUBST NIL NIL l), mais est evidemment beaucoup plus rapide. L'interprete se permet d'utiliser cette fonction a la place de la fonction SUBST dans le cas ou les deux premiers arguments de la fonction SUBST sont les memes pointeurs physiques.

(DCONS s l) [SUBR a 2 arguments] pour "Distributive CONS"

est equivalente a l'evaluation de :
(MAPCAR l (FUNCTION (LAMBDA (L) (CONS s l))))

(DDT) [SUBR a 0 argument]

permet d'appeller le programme DDT pour la mise au point des fonctions ecrites en LAP ou de l'interprete lui-meme. Si DDT n'a pas ete charge au moment du link-edit, cette fonction ramene NIL, sinon vous vous retrouvez sous DDT. Pour revenir a LISP, faites-lui executer RDDT$X.

(DELQ a l) [SUBR a 2 arguments]
(DELETE s l) [SUBR a 2 arguments]

ramene une copie du top-level de l dans laquelle toutes les occurences de l'atome a (pour la fonction DELQ) ou de la liste s (pour la fonction DELETE) ont ete enlevees.

ex : (DELQ 'A '(Z A Y A X A)) -> (Z Y X)

(FREVERSE l) [SUBR a 1 argument] pour "Fast REVERSE"

renverse physiquement et rapidement la liste l. Cette fonction doit etre manipulee avec precaution car elle modifie physiquement des structures LISPs, mais elle est evidement beaucoup plus rapide que la fonction traditionnelle REVERSE.

```
ex :  (SETQ L1 '(A B C D E)) -> (A B C D E)
      (SETQ L2 (CDR L1))     -> (B C D E)
      (SETQ L3 (LAST L1))    -> (E)
      (FREVERSE L1)          -> (E D C B A)
      L2                     -> (B A)
      L3                     -> (E D C B A)
```

(FUNCTION s) [FSUBR]

Cette fonction est identique a la fonction QUOTE pour l'interprete mais signale au compilateur qu'il doit compiler l'expression donnee en argument comme une fonction. Cette fonction est la seule "declaration" utilisee par le compilateur et n'est pas obligatoire. FUNCTION est principalement utilise dans les fonctionnelles.

(IFN e1 e2 e3 ... eN) [FSUBR] pour "IF Not"

Cette fonction est equivalente a :

```
        (IF (NOT e1) e2 e3... eN)
```

(INUMBP) [SUBR a 1 argument]

Ce nouveau predicat teste si l'argument, numerique, est un "petit" entier i.e. un nombre ayant une representation unique et se trouvant dans la zone speciale reservee a cet effet. La taille de cette zone est fixee a la generation du systeme. Ce predicat n'est utile que pour le compilateur.

(MCONS s1 s2 ... sN-1 sN) [SUBR a N arguments] pour "Multiple CONS"

Cette fonction est equivalente a :

```
        (CONS s1 (CONS s2 ( .... (CONS sN-1 sN) ... )))
        ex:  (MCONS 'A 'B 'C) -> (A B . C)
```

(NCONS s) [SUBR a 1 argument]

est equivalent a l'appel : (CONS s NIL). Cette fonction n'est pas tres utile pour vous mais est utilisee par le compilateur, pour ameliorer le code genere.

(SAMEPN a1 a2) [SUBR a 2 arguments]

ramene T si le P-name de l'atome a1 commence par le P-name de l'atome a2, sinon ce predicat ramene NIL. SAMEPN est utilise pour tester les premiers caracteres du P-name d'un atome.

```
ex : (SAMEPN '*L101 '* )    ->  T
     (SAMEPN '*L101 '*L)    ->  T
     (SAMEPN '*L101 '*M)    ->  NIL
```

(XCONS s1 s2) [SUBR a 2 arguments] pour "eXchange CONS"

est equivalent a l'appel : (CONS s2 s1). Cette fonction n'est pas tres utile pour vos fonctions interpretees mais elle permet d'ameliorer le code genere du compilateur.

4.3 Les fonctions testant les types

Du fait de l'introduction de nouveaux types, les fonctions de recherche sur type ont ete modifiees.

(TYPEP s) [SUBR a 1 argument]

ramene differents atomes en fonction du type de s. Cette fonction peut etre utilisee pour construire des fonctions "generic". TYPEP ramene l'atome :
- LITATOM si s est un atome litteral,
- NUMBP si s est un nombre (pour connaitre le type de ce nombre, il faut utiliser la fonction TYPNUMB),
- STRINGP si s est une chaine,
- LISTP si s est une liste,
- NIL si s n'est pas un objet LISP (par exemple une adresse de lancement de SUBR, une adresse de tableau etc ...).

(TYPEFN a) [SUBR a 1 argument]

Cette fonction teste si l'atome a donne en argument possede une definition de fonction. TYPEFN ramene l'atome :
- EXPR si a possede une definition de type EXPR,
- FEXPR si a possede une definition de type FEXPR,
- SUBR si a possede une definition de type SUBR,
- FSUBR si a possede une definition de type FSUBR,
- MACIN si a possede une definition de type MACIN,
- MACOUT si a possede une definition de type MACOUT,
- MACRO si a possede une definition de type MACRO,
- ARRAY si a possede une definition de type ARRAY,
- AUTOLOAD si a est une fonction de type AUTOLOAD,
- NIL si a ne possede pas de definition de fonction.

Si l'atome possede plusieurs definitions, TYPEFN ramene le type de celle qu'utiliserait EVAL si on `evaluait une forme dont la fonction etait a ,c'est-a-dire la premiere definition trouvee sur la P-liste de a.

(TYPNUMB n) [SUBR a 1 argument]

cette fonction teste le type du nombre donne en argument et ramene l'atome :
- FIX si le nombre n est de type entier,
- FLOAT si le nombre n est de type flottant,
- NIL si l'argument n n'est pas un nombre.

4.4 Les fonctions sur les P-listes

Les indicateurs sur les P-listes peuvent etre maintenant des listes. Toutes les fonctions de recherche sur les P-listes utilisent le predicat EQUAL. Toutefois pour accelerer la recherche, les fonctions GET GETL PUT et REMPROP testent d'abord si l'indicateur ne serait pas un atome litteral; en ce cas, une routine specialisee est employee. Cette routine utilise le predicat EQP et est extremement rapide. Une nouvelle fonction de recherche a ete ajoutee.

(GETL pl ind) [SUBR a 2 arguments]

pl est la P-liste a explorer et lind est une liste d'indicateurs. GETL ramene la sous-p-liste de pl commencant par un des element de lind. Cette fonction permet entre autre de lever l'ambiguite qui existait, dans la fonction GET, entre l'absence d'un indicateur et la valeur NIL associee a un indicateur.

```
ex : (GETL '(I1 1 I2 2 I3 3) '(I2 I4))  ->  (I2 2 I3 3)
     (GETL '(I1 1 I2 NIL) '(I2))  ->  (I2 NIL)
```

4.5 Les fonctions sur pile

La zone dans laquelle est stockee la pile utilisateur etant partagee avec les tableaux, la taille de cette pile est donc determinee par la difference entre la taille de la zone allouee aux tableaux et la somme des tailles de tous les tableaux definis. Les 2 fonctions sur pile ont subi des modifications.

(PUSH s1 ... sN) [SUBR a N arguments]

comme auparavant, PUSH empile les valeurs des differentes

expressions s1 ... sN et ramene sN en valeur. Toutefois si aucun argument n'est fourni (i.e. si on evalue (PUSH)), une valeur NIL est quand meme empilee.

(POP n) [SUBR a 1 argument]

si l'argument n'est pas fourni, l'effet est identique que precedement : le sommet de pile est ramene en valeur et le pointeur de pile est mis a jour. Si l'argument numerique n est donne, POP ramene le nieme element contenu dans la pile a partir du pointeur de pile mais celui-ci n'est pas modifie. Si ce nombre est negatif, on peut avoir acces a des elements de la pile precedemment empiles puis depiles. Il ne faut toutefois pas sortir de la zone allouee pour la pile sous peine de declencher les erreurs de debordement de pile.

```
ex : (PUSH 'A 3)        -> 3
     (PUSH)             -> NIL
     (PUSH 'B)          -> B
     (POP 0)            -> B
     (POP 1)            -> NIL
     (POP 2)            -> 3
     (POP)              -> B
     (POP)              -> NIL
     (POP 1)            -> A
     (POP -2)           -> B
     (POP)              -> 3
           ... etc ...
```

SECTION 5

LES ENTREES-SORTIES

Les atomes litteraux ont un P-name de 13 caracteres au maximum (au lieu de 18). Il est preferable d'utiliser les chaines de caracteres pour imprimer des libelles ou du texte.

5.1 La fonction READ

La recherche des atomes dans l'OBLIST a ete acceleree en replacant notamment les atomes en tete de l'OBLIST a chacune de leur apparition.

Deux macros-caracteres standards ont ete introduits en entree : le crochet ouvrant [et le crochet fermant]. Ils correspondent a l'appel de la fonction LIST. L'ecriture [e1 e2 ... eN] est maintenant equivalente a (LIST e1 e2 ... eN).

Un transcodage des caracteres minuscules en leurs equivalents majuscules est effectue, sauf pendant la lecture des chaines de caracteres et des commentaires. Les S-expressions Lisp peuvent donc etre tapees indifferemment en majuscules ou en minuscules.

Les parentheses fermantes) en exces a la fin d'une expression sont ignorees. Ceci permet de prendre la tres mauvaise habitude de terminer la frappe des S-expressions par un grand nombre de parentheses fermantes.

Les caracteres numeriques se trouvant dans des chaines de caracteres ou dans les P-names des atomes sont consideres comme des nombres.

```
   ex : (ADD1 (CADR (EXPLODE 'A4L)))  -> 5
```

Le delimiteur de paire pointee (.) doit etre maintenant encadree du caractere espace car le caractere point est egalement utilise pour l'ecriture des nombres flottants.

```
   ex : '(VAR . VAL)      et non : '(VAL.VAR)
```

L'erreur ** READ error. apparait toujours en cas de mauvaise utilisation du delimiteur de paire pointee (.) et egalement si des crochets carres [ ] ne se ferment pas exactement comme par exemple dans l'expression [(A (B)].

La fin d'un fichier d'entree est toujours signalee par le message ** E.O.F. emis par la fonction standard EOF. Toutefois si une S-expression etait en cours de lecture (non terminee) le message devient ** E.O.F. during READ.

5.2 La fonction PRINT

La restitution du macro-caractere QUOTE (') en sortie ne se fait qu'a l'occurence de la fonction QUOTE a 1 argument : (QUOTE A B) ne donne plus 'A (ce qui etait erronne) mais (QUOTE A B).

L'impression d'un objet non-lisp (du genre adresse de lancement de SUBR) est faite en octal, prefixee par le caractere \ (qui est le macro- caractere de passage en mode octal), ce qui en assure la relecture.

```
ex: (VAG 'CAR) -> \14000000
```

5.3 Les speficicactions de fichier

La forme generale d'une specification de fichier est deumeuree inchangee :
(device (filename . extension) (projet . programmeur))

```
device      : les 3 caracteres du nom du peripherique
filename    : nom du fichier (6 caracteres au maximum)
extension   : nom de l'extension (3 caracteres au maximum)
projet      : numero (ou nom) de projet
programmeur : numero (ou nom) du programmeur.
```

L'ensemble (projet.programmeur) est la specification d'un repertoire (ppn). L'absence de cette specification implique l'utilisation du repertoire de l'utilisateur. Les repertoires de type IRCAM (i.e. pouvant contenir des lettres) sont correctement traites.

```
Specifications par defaut :
device    : TTY (le terminal)
filename  : en entree LISPIN, en sortie LISPOU
extension : en entree VLI (pour un fichier ecrit en VLISP)
                LAP (pour un fichier issu du compilateur)
                LOD (pour un fichier issu de l'assembleur)
            en sortie  LST (pour les sorties normales)
                       PRT (pour les sorties issues
                            du PRETTY-PRINT)
projet.programmeur : celui de l'utilisateur
```

```
Par defaut donc,
(INPUT)          == (INPUT '(TTY (LISPIN . VLI) () ))
(OUTPUT)         == (OUTPUT '(TTY (LISPOU . LST) () ))
```

Une nouvelle ecriture allegee est egalement possible, constituee d'un atome simple; elle correspond a la forme :
(DSK (atome . extension standard) (utilisatueur))

```
(INPUT 'FOO)     == (INPUT '(DSK (FOO . VLI) () ))
(OUTPUT 'FOO)    == (OUTPUT '(DSK (FOO . LST) () ))
```

5.4 Fonctions sur les fichiers

Tous les fichiers d'entree sont compatibles avec l'editeur de l'IRCAM ETV. La premiere page du fichier, qui contient son repertoire et qui est mis a jour par l'editeur, est consideree par la fonction d'entree READ comme un commentaire. IL faut toutefois prendre garde a ne pas mettre un caractere C en tete d'un fichier; ce caractere C etant justement la marque du repertoire de ETV.
Un nouveau mode de chargement rapide a ete ajoute : le mode LIBRARY, qui, silencieusement, vous charge n'importe quel fichier. Ce fichier (d'extension VLI, LAP ou LOD) peut se trouver dans votre zone disque ou bien dans d'autres zones que vous pouvez specifier grace a une nouvelle fonction PATHLIBRARY.

(LIBRARY file) [FSUBR]

charge le fichier file sans impression. Ce fichier doit posseder l'une des extensions standards VLI LAP ou LOD. Si le fichier existe et si le chargement s'est correctement effectue, cette fonction ramene file i.e. le nom de ce fichier. Danc tous les autres cas LIBRARY ramene NIL. Le repertoire disque dans lequel le fichier doit etre cherche est precise par la fonction suivante.

```
ex : (OR (EQ (TYPEFN 'PHENARETES) 'EXPR)
         (LIBRARY PHENAR)
         (ERROR "mais ou est-elle donc passee ?"))
```

(PATHLIBRARY ppn1 ppn2 ... ppnN) [SUBR a N arguments]

indique a la fonction LIBRARY les differents repertoires dans lesquels elle doit rechercher un fichier. Le repertoire utilisateur est indique par NIL. La recherche s'effectue dans l'ordre des arguments donnes a cette fonction.
Par defaut (PATHLIBRARY () SYS), indique qu'il faut d'abord essayer de chercher le fichier dans le repertoire de l'utilisateur et en cas d'echec dans celui du systeme.

(DIRECTORY ppn) [SUBR a 1 argument]

ramene la liste de tous les fichiers du repertoire ppn. Cette fonction est utilisee pour s'assurer de la presence de fichier en restant sous LISP.

5.5 Les fichiers image-memoire

Deux nouvelles fonctions ont ete crees pour pouvoir sauver l'image memoire de votre zone de travail sur disque. Ces deux nouvelles fonctions deviendront par la suite inutiles a la mise en service de la nouvelle UUO IRCAM SWAP. Si vous ne precisez pas de fichier pour ces deux fonctions, le fichier par defaut sera :
(DSK (TEMPOR . COR) ()) .

(WRCORE filespec) [SUBR a 1 argument]

ecrit dans le fichier specifie votre image memoire. Cette fonction apres execution retourne au top-level de VLISP.

(RDCORE filespec) [SUBR a 1 argument]

restaure votre image memoire a partir du fichier specifie. Comme WRCORE cette fonction retourne apres execution au top-level de VLISP.

SECTION 6

LE PRETTY-PRINT

6.1 Introduction

Les fonctions standards PRINT et PRIN1 sont d'ordinaire utilisees pour editer les S-expressions LISPs. Les seules mesures prises pour ameliorer la lisibilite sont :
- l'insertion d'un espace entre chaque atome ;
- l'interdiction d'editer un atome sur deux lignes.

Ces mesures sont nettement insuffisantes pour editer vos programmes. Les fonctions du PRETTY-PRINT vont les editer d'une maniere beaucoup plus lisible en faisant resortir, au moyen de renfoncements gauches et de sauts de lignes ad hoc, la structure de controle de vos fonctions. Les macros-caracteres definis de maniere standard dans le systeme (i.e. les caracteres ' [ ]) sont egalement restitues par le PRETTY-PRINT.
Les fonctions standards qui vont etre decrites sont de type AUTOLOAD i.e. il n'est pas besoin de charger le fichier qui les contient, le systeme le fera pour vous au premier appel de l'une de ces fonctions. Si malgre tout il vous semble utile de charger ce fichier par vous-meme, sachez qu'il se nomme PRETTY et qu'il est aisement charge par l'evaluation de (LIBRARY PRETTY).

6.2 Les Fonctions du PRETTY-PRINT

(PRETTYP s) [SUBR a 1 argument]

edite la S-expression s. Cette fonction ramene s en valeur. Elle est utilise en general dans vos propres fonctions qui doivent editer des objets LISPs.

(PRETTY at1 at2 ... atN) [FSUBR]

edite les definitions existantes de type EXPR, FEXPR ou MACRO des differents atomes at1 ... atN. Cette fonction ramene toujours NIL en valeur. Elle est utilisee en mise au point conversationnelle, pour verifier aisement un parenthesage douteux.

(PRETTYFILE filout filin1 ... filinN) [SUBR a N arguments]

edite dans le fichier de sortie filout, toutes les S-expressions contenues dans les differents fichiers d'entree filin1 ... filnN. Cette fonction permet de faire des copies lisibles de vos programmes apres mise au point. S'il n'y a pas de specifications pour le fichier de sortie (filout = NIL), le fichier (DSK(PRETTY.PRT)) est utilise. L'extension standard des fichiers de sortie du PRETTY-PRINT est .PRT . Ces fichiers peuvent etre relus par LISP, mais doivent etre reformates pour utiliser l'editeur de l'IRCAM ETV.
Cette fonction a le bon gout de lire et de reecrire les commentaires (i.e. la suite de caracteres quelconques encadree du delimiteur ";") qui sont d'habitude completement ignores de LISP. Ceci devrait vous donner l'envie de bien commenter les programmes.
Un saut de page est effectue a chaque occurence du caractere ^L (Form Feed) ou a la rencontre du commentaire speciale ;PAGE; .

(PRETTYF file) [FSUBR]

est une forme abregee de la fonction PRETTYFILE. cette fonction correspond a l'appel suivant :
(PRETTYFILE '(DSK (file . PRT)) '(DSK (file . VLI)))

(PRETTYSIZE n) [SUBR a 1 argument]

permet d'initialiser la largeur d'impression du PRETTY-PRINT. Par defaut cette largeur est de 72 ce qui correspond a-peu-pres au format standard 21x29,7. Si n n'est pas donne ou n'est pas un nombre, la largeur standard est prise.

(PRETTYEND) [SUBR a 0 argument]

permet de recuperer la place occupee par les fonctions du PRETTY-PRINT. PRETTYEND remet egalement les indicateurs AUTOLOADs de ces fonctions. PRETTYEND ramene en valeur le nombre de doublets liberes. Si vous utilisez le PRETTY-PRINT compile, cette fonction n'a aucun effet.

SECTION 7

LE LAP

LAP est un assembleur (ressemblant a l'assembleur du PDP10), concu pour etre utilise par VLISP-10.2, prevu originellement pour charger le code issu du compilateur. Il peut etre utilise seul pour ecrire de nouvelles fonctions standards. Un minimum de connaissances de l'assembleur PDP10 et de l'organisation de l'interprete est requis pour utiliser le LAP sans degats.
LAP recoit en argument une liste dont les elements peuvent etre :
- des atomes litteraux (qui servent d'etiquettes)
- des listes representant des instructions normales, des pseudos-instructions, ou des macros-LAPs (MACLAPs).
Cette liste peut etre remplacee par un fichier s'il y a beaucoup d'instructions a charger.

7.1 Les registres

L'interprete utilise les 16 registres de la machine de la maniere suivante :

```
no symbol       remarques

00 RG    est le registre general lui-meme. On peut donc tester
         directement au moyen d'un masque les bits utiles.

01 A1    accumulateur 1   !
02 A2    accumulateur 2   !  Seuls ces 4 registres sont nettoyes par
03 A3    accumulateur 3   !  le G.C.
04 A4    accumulateur 4   !

05 A5    accumulateur 5
06 A6    accumulateur 6    Ces 4 registres font office de registres
07 A7    accumulateur 7    auxiliaires.
10 A8    accumulateur 8

11 U1    user 1           !  Ces 2 registres ne sont
12 U2    user 2           ! jamais utilises par l'interprete.

13 L     link     est utilise pour les appels de S.P
                  de type JSP.

14 STRG           contient toujours le pointeur sur la liste
                  libre des chaines.

15 NUMB           contient toujours le pointeur sur la liste
                  libre des nombres.

16 FREE           contient toujours le pointeur sur la liste
```

libre des listes.

17 P est le pointeur sur l'unique pile utilisee par l'interprete.

7.2 Acces aux objets LISPs

Les atomes litteraux, les nombres, les chaines et les listes sont stockees dans des zones fixes du LOWSEG. Ces objets sont toujours representes d'une maniere interne par un -pointeur- sur ces zones ; ce pointeur n'est pas une adresse physique mais un index par-rapport au debut de cette zone. On a acces aux limites de ces zones au moyen de symboles speciaux (dont le 1er caractere est un ":") qui sont connus du LAP.

:MEM adresse physique du debut de la zone des objets LISPs.

- ZONE DES ATOMES LITTERAUX -

:BNUMB index du debut de la zone des nombres.

- ZONE DES NOMBRES -

:BSTRG index du debut de la zone des chaines.

- ZONE DES CHAINES -

:BLIST index du debut de la zone des listes.

- ZONE DES LISTES -

7.3 Test de type

Grace a ce decoupase de la memoire, le test de type se ramene a une comparaison avec les limites des zones, ce qui est tres efficient. Le 1er atome stocke dans la zone des atomes litteraux est l'atome NIL ; la valeur de son index est 0 ; les tests par-rapport a NIL peuvent se coder en une seule instruction (JUMPE ou JUMPN).

ex : branchement si A1 est un nombre

```
(CAML A1 :BNUMB)
(CAML A1 :BSTRG)
(JRST 0 quelquepart)
```

7.4 Creation d'objets

Pour creer un atome litteral, il faut preparer dans un petit buffer (de 3 mots) de nom :PNAME (ce nom est connu de LAP) le P-name de l'atome que l'on veut creer puis appeller une routine de l'interprete de nom :TRYATOM au moyen d'un (PUSHJ P :TRYATOM). Ce buffer doit contenir les caracteres de P-name en code ASCII. Le 1er caractere doit etre le nombre total de caracteres du P-name. Ce buffer est complete avec des 0. Au retour le registre A1 contient le pointeur sur ce nouvel atome.

Pour creer un nombre, il faut mettre sa valeur dans le registre A5 puis appeller la routine :CRANUM pour creer un nombre entier ou bien la routine :CRAFLT pour un nombre flottant. Au retour le registre A1 contient le pointeur sur ce nouveau nombre.

Pour creer une chaine, il faut mettre la liste contenant tous les caracteres (sous forme atomique) dans le registre A1 puis appeller la routine de l'interprete :CRASTR au moyen d'un (PUSHJ P :CRASTR). Au retour le registre A1 contient le pointeur sur cette nouvelle chaine.

On peut creer un doublet de liste directement. Il faut preparer un registre quelconque (A1 A2 A3 ou A4) avec en partie gauche le CAR du doublet et en partie droite son CDR puis tester si la liste libre des doublets (pointee par le registre FREE) n'est pas vide (auquel cas il faut appeller le "Garbage-Collecting") et enfin stocker le doublet et actualiser FREE. Ces operations peuvent s'ecrire :

```
          ; preparation du registre A1
     (JUMPN FREE (* 2))       ; si FREE est vide,
     (PUSHJ P :GARBCL)    ; appeller le G.C.
     (EXCH A1 :MEM FREE)      ; creation du doublet
     (EXCH FREE A1)           ; actualisation de FREE
          ; le pointeur obtenu se trouve dans A1
```

7.5 Format externe d'une instruction

Chaque instruction est representee par une liste de la forme :

(<codop> <registre> @ <adr> <index>)

<codop> - est le mnemonique de l'instruction (tous les mnemoniques PDP10 sont disponibles.
<registre> - est le numero ou le symbole du registre 1er operande.
@ - si cet atome est present (a cette position), le bit d'indirection de l'instruction sera positionne.
<adr> - represente la valeur du champ adresse qui sera charge dans les 18 bits de poids faibles de l'instruction (la description de ce champ suit).
<index> - est le numero ou le symbole du registe d'index. Ce dernier champ est otionnel.

le champ <adr> peut avoir la representation suivante :

- un nombre (utilise dans les intructions possedant des valeurs immediates).
- un atome litteral qui peut etre :
  - un symbole de registre
  - une etiquette
  - un symbole special. LAP connait en effet certaines adresses utiles de l'interprete. Elles sont representees par un symbole dont le 1er caractere est toujours un ":".
- une liste representant :
  - (nombre) une adresse relative au debut de chargement.
  - (* nombre) une adresse relative au compteur d'assemblage.
  - (QUOTE objet lisp) ou ' objet lisp , une adresse de l'objet lisp specifie. Cette adresse est en realite l'index de cet objet par rapport au debut de la zone des objets lisps.
  - (+ <adr> ... <adr>) la somme des differentes adresses specifiees.

ex d'instruction :

```
(MOVEI A1 2)
(MOVE A1 @ TAB A5)
(CAML A4 :BLIST)
(JRST 0 (* -5))
(CAIE A4 'LAMBDA)
(HRRZ A3 (+ :MEM 'X))
```

7.6 Les pseudos-instructions

LAP connait un certain nombre de pseudos-instructions dont la forme externe est identique a celle des instruction normales.

(EXP <adr>)

reserve un mot qui est initialise avec la valeur de <adr>.
ex : (EXP -1)

(XWD <adr> <adr>)

reserve un mot dont les 2 demis-mots sont initialises avec les valeurs des operandes fournis.
ex : (XWD -1 (+ :MEM (* 3)))

(QUOTE objet lisp ou bien ' objet lisp)

reserve un mot contenant un pointeur sur l'objet lisp specifie.

(VALAP symbole valeur)

permet de definir de nouveaux symboles dont la valeur est donnee explicitement.
ex : (VALAP :JBSYM \116)

(OPCD symbole valeur)

permet de definir de nouveaux mnemoniques pour les codes instructions.

ex : (OPCD PJRST \254)

(ENTRY nom type nombre)
permet de definir le point d'entree d'une fonction qui devient une fonction standard de VLISP. Le nom de la fonction doit etre un atome litteral, le type de cette fonction peut etre SUBR ou FSUBR. Si le type est SUBR on peut specifier son nombre d'arguments (si cette fonction possede un nombre d'arguments plus petit ou egal a 3).
ex : (ENTRY FACT SUBR 1)

(END)
indique la fin d'un assemblage (un 2ème peut suivre dans la liste des instructions fournie au LAP). Cette pseudo est automatiquement generee en fin de liste d'instruction ; elle n'est donc pas obligatoire.

7.7 Les macros-LAP

Il est possible de definir des macros en utilisant la pseudo suivante :

(MACLAP nom liste d'argument corps de la fonction)

cette pseudo-instruction permet de definir les MACLAPs. A l'apparition d'une intruction de la forme (nom arg1 .. argN) dans laquelle nom est le nom d'une MACLAP, la fonction specifiee est appellee avec arg1 ... argN comme arguments. C'est la valeur ramenee par cette application qui est assemblee (si cette valeur est differente de NIL).

exemple de definition de MACLAP :

```
(MACLAP INCR (ATOM)
        [['HLRZ 1 ['+ ':MEM [QUOTE ATOM]]]
         '(PUSHJ P ADD1)
         ['HRLM 1 ['+ ':MEM [QUOTE ATOM]]]]])
```

```
l'appel de  (INCR X)
sera expanse en  (HLRZ 1 (+ :MEM 'X))
                 (PUSHJ P ADD1)
                 (HRLM 1 (+ :MEM 'X))
```

Les MACLAPs sont tres puissantes car elles donnent acces a l'interprete lui-meme.

Il existe un certain nombre de MACLAP predefinies qui possedent les definitions suivantes :

```
CAR          (MACLAP CAR (D S) [['HLRZ D ':MEM S]])
```

```
CDR          (MACLAP CDR (D S) [['HRRZ D ':MEM S]])

JPLIST       (MACLAP JPLIST (R A)
                         [['CAMGE R ':BLIST]
                          ['JRST 0 A]])

JNLIST       (MACLAP JNLIST (R A)
                         [['CAML R ':BLIST]
                          ['JRST 0 A]])

CONS         (MACLAP CONS (R)
                         [ '(JUMPN FREE (* 2))
                           '(PUSHJ P :GARBCL)
                           ['EXCH R ':MEM 'FREE]
                           ['EXCH 'FREE R]])

UNCONS       (MACLAP UNCONS (R CAR CDR)
                    (IF (NEQ R CDR)
                         [['HRRZ CDR ':MEM R]
                          ['HLRZ CAR ':MEM R]]
                         [['HLRZ CAR ':MEM R]
                          ['HRRZ CDR ':MEM R]]))
```

7.8 Appel des fonctions

Toutes les fonctions de l'interprete sont appellees au moyen d'un PUSHJ P,adresse de lancement de la fonction. Toutes les fonctions doivent donc se terminer par un POPJ P. Les arguments des fonctions sont transmis dans differents registres en fonction du type de la fonction. Cette transmission est effectuee automatiquement, avant l'appel des fonctions, par les fonctions interprete EVAL ou APPLY. Quelque soit le type d'une fonction, sa valeur est toujours retournee dans le registre A1 (il faut donc que ce registre soit charge avant le retour de la fonction). Les fonctions de type SUBRs a 1 argument recoivent leur argument evalue dans le registre A1, les SUBRs a 2 arguments recoivent leurs arguments evalues dans les registres A1 (pour le 1er) et A2 (pour le 2eme), les SUBRs a 3 arguments recoivent leurs argments evalues dans les registres A1 (pour le 1er) A2 (pour le 2eme) et A3 (pour le 3eme), les SUBRs a nombre quelconque d'arguments recoivent une liste contenant tous les arguments evalues dans le registre A4, les fonctions de type FSUBRs recoivent la liste des arguments non-evalues dans le registre A1.

7.9 Les Fonctions standards du LAP

Les deux fonctions du LAP sont de type AUTOLOAD et se trouvent sur un fichier de nom LODLAP.

(LAP l sw1 sw2) [SUBR a 3 arguments]

est la principale fonction du LAP. Le premier argument l doit etre une liste d'instructions LAP; le deuxieme argument est un indicateur qui vaut T si vous desirez une trace de ses cogitations; le troisieme est l'indicateur qui sera transmis au chargeur en cas de besoin.

(LAPF file sw1 sw2) [SUBR a 3 arguments]

cette fonction travaille d'une maniere identique a la fonction LAP, toutefois la liste des instructions LAP se trouve dans un fichier dont la specification est donne comme premier argument. LAPF est utilise lorsque la liste des instructions a fournir au LAP est tres grande. Ce fichier ne doit contenir que des instructions LAPs.

(LAPEND) [SUBR a 0 argument]

permet de recuperer la place occupee par les fonctions du LAP, ainsi que les indicateurs places sur certains atomes par le LAP. Si certains atomes contiennent toujours des references non resolues (ces atomes possedent sur leur P-liste l'indicateur *UDS), un avertissemnet est donne sous la forme :
** undefined symbol : le nom de la fonction toujours inconnue.

7.10 Exemples d'utilisation du LAP

```
    ; definition de la fonction ONEP      ;
    ;        equivalent LISP :            ;
    ;  (DE ONEP (X) (EQ X 1))             ;

?   (LAP '((ENTRY ONEP SUBR 1) (CAIE A1 '1) (TDZA A1 A1)
?          (MOVEI A1 'T) (POPJ P)) () T)

    51742                           (ENTRY ONEP SUBR 1)
    51742   302 1   0 0   13661     (CAIE A1 '1)
    51743   634 1   0 0   1         (TDZA A1 A1)
    51744   201 1   0 0   6         (MOVEI A1 'T)
    51745   263 17  0 0   0         (POPJ P)
    51746
```

```
   (51742 ONEP SUBR 1)
                                       (END)
=  LAP
=  ; time = 220 ms ;

?  (TYPEFN 'ONEP)
=  SUBR
=  ; time = 0 ms ;

?  (ONEP 1)
=  T
=  ; time = 0 ms ;

?  (ONEP 0)
=  NIL
=  ; time = 0 ms ;

?  (ONEP)
=  NIL
=  ; time = 0 ms ;

  ;   definition des fonctions :         ;
  ; (DE.CADDDR (X) (CAR (CDDDR X)))      ;
  ; (DE CDDDDR (X) (CDR (CDDDR X)))      ;

?  (LAP '((ENTRY CADDDR SUBR 1) (SKIPA A1 :MEM A1)
?         (ENTRY CADDAR SUBR 1) (HLRZ A1 :MEM A1)
?         (JRST 0 CADDR)) NIL T)

   51746                               (ENTRY CADDDR SUBR 1)
   51746  334 1  0 1  2154             (SKIPA A1 :MEM A1)
   51747                               (ENTRY CADDAR SUBR 1)
   51747  554 1  0 1  2154             (HLRZ A1 :MEM A1)
   51750  254 0  0 0  405546           (JRST 0 CADDR)
   51751

   (51746 CADDDR SUBR 1)
   (51747 CADDAR SUBR 1)
                                       (END)
=  LAP
=  ; time = 200 ms ;

?  (SETQQ L ((A B C D) E F G H))
=  ((A B C D) E F G H)
=  ; time = 0 ms ;

?  (CADDAR L)
=  C
=  ; time = 0 ms ;

?  (CADDDR L)
=  G
=  ; time = 0 ms ;
```

```
   ; redefinition de la fonction REVERSE standard ;
   ; (DE REV (L1 L2)                                ;
   ;   (WHILE (LISTP L1)                            ;
   ;     (SETQ L2 (CONS (CAR L1) L2))               ;
   ;     (SETQ L1 (CDR L1)))                        ;
   ;   L2)                                          ;

?  (LAP '((* "REVERSE tres standard.")
?        RE (HLL 2 :MEM 1) (CONS 2)
?        (CDR 1 1) (ENTRY REV SUBR 2) (JPLIST 1 RE)
?        (MOVEI 1 0 2) (POPJ P)) T T)

   0           (* "REVERSE tres standard.")
   0     RE
   0           (HLL 2 :MEM 1)
   1           (CONS 2)
   5           (CDR 1 1)
   6           (ENTRY REV SUBR 2)
   6           (JPLIST 1 RE)
   10          (MOVEI 1 0 2)
   11          (POPJ P)

   51751  = 51751               (VALAP RE (0))
   51751                        (* "REVERSE tres standard.")
   51751  500 2  0 1  2154      (HLL 2 :MEM 1)
   51752  326 16 0 0  51754     (JUMPN FREE (* 2))
   51753  260 17 0 0  400362    (PUSHJ P :GARBCL)
   51754  250 2  0 16 2154      (EXCH 2 :MEM FREE)
   51755  250 16 0 0  2         (EXCH FREE 2)
   51756  550 1  0 1  2154      (HRRZ 1 :MEM 1)
   51757                        (ENTRY REV SUBR 2)
   51757  311 1  0 0  1260      (CAML 1 :BLIST)
   51760  254 0  0 0  51751     (JRST 0 RE)
   51761  201 1  0 2  0         (MOVEI 1 0 2)
   51762  263 17 0 0  0         (POPJ P)
   51763

   (51757 REV SUBR 2)
                                (END)
=  LAP
=  ; time = 460 ms ;

?  (REV '(A (B C) D . E) '(F G H))
=  (D (B C) A F G H)
=  ; time = 0 ms ;

   ; definition de la fonction FIND qui pourrait  ;
   ; se definir en VLISP de la maniere suivante : ;
   ;                                              ;
   ; (DE FIND (A L)                               ;
   ;    (ESCAPE EXIT (FIND1 L)))                  ;
   ;                                              ;
   ; (DE FIND1 (L)                                ;
   ;    (IF (ATOM L)                              ;
```

```
;       (IF (EQ A L) (EXIT A))                      ;
;      (FIND1 (CAR L))                              ;
;      (FIND1 (CDR L))))                            ;

?  (LAP '((*) (* "(FIND A L)") (*) (ENTRY FIND SUBR 2)
?        (MOVEM P SAVEP) (PUSHJ P FIND2) (SETZ 1) (POPJ P)
?        RE (UNCONS 2 2 3) (PUSH P 3) (PUSHJ P FIND2) (POP P 2)
?        FIND2 (JPLIST 2 RE) (CAIN 1 0 2) (MOVE P SAVEP)
?        (POPJ P) SAVEP (EXP 0)) T T)

 0              (*)
 0              (* "(FIND A L)")
 0              (*)
 0              (ENTRY FIND SUBR 2)
 0              (MOVEM P SAVEP)
 1              (PUSHJ P FIND2)
 2              (SETZ 1)
 3              (POPJ P)
 4      RE
 4              (UNCONS 2 2 3)
 6              (PUSH P 3)
 7              (PUSHJ P FIND2)
 10             (POP P 2)
 11     FIND2
 11             (JPLIST 2 RE)
 13             (CAIN 1 0 2)
 14             (MOVE P SAVEP)
 15             (POPJ P)
 16     SAVEP
 16             (EXP 0)

 51763  = 52001                 (VALAP SAVEP (16))
 51763  = 51774                 (VALAP FIND2 (11))
 51763  = 51767                 (VALAP RE (4))
 51763                          (*)
 51763                          (* "(FIND A L)")
 51763                          (*)
 51763                          (ENTRY FIND SUBR 2)
 51763  202 17 0 0   52001      (MOVEM P SAVEP)
 51764  260 17 0 0   51774      (PUSHJ P FIND2)
 51765  400 1  0 0   0          (SETZ 1)
 51766  263 17 0 0   0          (POPJ P)
 51767  550 3  0 2   2154       (HRRZ 3 :MEM 2)
 51770  554 2  0 2   2154       (HLRZ 2 :MEM 2)
 51771  261 17 0 0   3          (PUSH P 3)
 51772  260 17 0 0   51774      (PUSHJ P FIND2)
 51773  262 17 0 0   2          (POP P 2)
 51774  311 2  0 0   1260       (CAML 2 :BLIST)
 51775  254 0  0 0   51767      (JRST 0 RE)
 51776  306 1  0 2   0          (CAIN 1 0 2)
 51777  200 17 0 0   52001      (MOVE P SAVEP)
 52000  263 17 0 0   0          (POPJ P)
 52001  0                       (EXP 0)
 52002
```

```
        (51763 FIND SUBR 2)
                                          (END)
=    LAP
=    ; time = 460 ms ;

?    (FIND 'A 'A)
=    A
=    ; time = 0 ms ;

?    (FIND 'A 'B)
=    NIL
=    ; time = 0 ms ;

?    (FIND 'A '((B (C (D A . Z)))))
=    A
=    ; time = 0 ms ;

?    (FIND 'A '(B (C D E) F))
=    NIL
=    ; time = 0 ms ;
```

SECTION 8

LE COMPILATEUR

VLISP-10 possede maintenant un compilateur. Il est sans danger et peut etre utilise par tous. Ce compilateur peut traduire n'importe quelle fonction definie de type EXPR ou FEXPR en une suite d'instructions qui sera transmise au LAP. Il n'utilise pas de declaraions speciales et suppose que les fonctions que vous voulez compiler sont -correctes- . Les fonctions ainsi compilees sont executees entre 3 et 10 fois plus rapidement, et occupe moins de place (avec un gain variant entre 20 et 200 %). Ce compilateur garanti l'identite de resultat avec la forme interpretee sauf en cas d'automodification de fonction ou en cas de redefinition dynamique de fonctions standards.

8.1 Les macros du compilateur

Le compilateur utilise un certains nombre de macros-LAP, connues du LAP qui facilite son ecriture et font gagner de la place a la generation du code.

(GETVAL registre atome)
    charge dans le registre specifie la C-valeur de l'atome. Cette macro est expansee en :
        (HLRZ registre [+ :MEM 'atome])

(PUTVAL registre atome)
    met dans la C-valeur de l'atome specifie, la valeur du registre. Cette macro est expansee en :
        (HRLM registre [+ :MEM 'atome])

(SETNIL atome)
    met NIL dans la C-valeur de l'atome specifie. Cette macro est expansee en :
        (HRRZS 0 [+ :MEM 'atome])

(CAR regd source)
    met le CAR de l'objet source dans le registre regd. Cette macro est expansee en :
    si l'objet source est
    - un registre (HLRZ regd :MEM source)
    - un atome (HLRZ regd [+ :MEM 'atome])
        cette forme est equivalente au GETVAL.

(CDR regd source)
    met le CDR de l'objet source dans le registre regd. Cette macro est expansee en :
    si l'objet source est

- un registre (HRRZ regd :MEM registre)
- un atome (HRRZ regd [+ :MEM 'aome])

8.2 Points d'entree speciaux

Le compilateur utilise un certain nombre de points d'entree speciaux dans l'interprete. Les noms de ces points d'entree sont toujours prefixes par le caractere ":" et sont bien evidemment connus de LAP.

:VPOPJ
est l'adresse d'une instruction (POPJ P)

:TRUTH
est l'adresse des retours vrais des predicats. Elle correspond aux deux instructions : (MOVEI 1 'T) (POPJ P).

:FALSE
est l'adresse des retours faux des predicats. Elle correspond aux deux instructions : (SETZ 1) (POPJ P).

:CRAZER
est l'adresse de creation du nombre LISP 0. Elle correspond aux deux instructions : (MOVEI 1 '0) (POPJ P).

:CRAONE
est l'adresse de creation du nombre LISP 1. Elle correspond aux deux instructions : (MOVEI 1 '1) (POPJ P).

:PLUS
est l'adresse de la fonction PLUS a deux arguments qui doivent se trouver respectivement dans les registres 1 et 2. Ce point d'entree est utilise quand la fonction PLUS n'est pas utilisee en tant que NSUBR.

:DIFFER
est l'equivalent de l'adresse :PLUS, pour la NSUBR DIFFER.

:TIMES
est l'equivalent de l'adresse :PLUS, pour la NSUBR TIMES.

:QUO
est l'equivalent de l'adresse :PLUS, pour la NSUBR QUO.

:MAPC1 :MAPCN
ces deux points d'entree sont utilises pour executer la fonction MAPC sans passer par APPLY. Ils peuvent etre utilises quand le compilateur connait le type de la fonction a appliquer aux elements successifs de la liste. Ces fonctions peuvent etre soit des fonctions standards, soit des fonctions compilees de

type FUNCTION. Si cette fonction est du type 1SUBR le point d'entree utilise est :MAPC1 sinon le compilateur utilise le point d'entree :MAPCN. L'execution de ce type de MAPC est fortement accelere par cette possibilite. Ces deux points d'entree attendent la liste dans le registre 1 et le nom de la fonction dans le registre 2.

:NSUBR
ce point d'entree est utilise pour lancer les NSUBRs de l'interprete ou bien les votres. Les NSUBRs demandent que leurs arguments, evalues, soient rassembles dans une liste qui doit se trouver dans le registre 4. Ceci est fait automatiquement si la pile est chargee de la maniere suivante : Il faut empiler d'abord un mot contenant dans sa partie gauche la valeur -1 et dans sa partie droite le nom de la NSUBR a lancer. Les arguments sont alors evalues un a un et leurs valeurs sont empilees sauf le dernier qui reste dans le registre 1. Puis le sous-programme :NSUBR est appelle via le registre L, au moyen d'un JSP.

ex: macrogeneration de l'expression
(TIMES 7 (STATUS 4 3 2) 8)

```
(PUSH  P :T1)      ; empile (XWD -1 TIMES) ;
(PUSH  P :T2)      ; empile la valeur 7 ;
(PUSH  P :T3)      ; empile (XWD -1 STATUS) ;
(PUSH  P :T4)      ; empile la valeur 4 ;
(PUSH  P :T5)      ; empile la valeur 3 ;
(MOVEI 1 '2)       ; le dernier argument
                     de STATUS ;
(JSP   L :NSUBR)   ; appel de STATUS ;
(MOVEI 1 '8)       ; le dernier argument
                     de TIMES ;
(JSP   L :NSUBR)   ; appel de TIMES ;
                   ; Table utilisee car le PDP-10
                     ne possede pas l'instruction
                     PUSH immediat !! ;
:T1 (XWD  -1 TIMES)
:T2 '7
:T3 (XWD  -1 STATUS)
:T4 '4
:T5 '3
```

Ce genre d'appel est plutot lourd et emcombrant, le compilateur essaie d'eviter cela dans le cas des NSUBRs utilisees frequemment et qui n'ont qu'un ou deux arguments.

:NSUBRP
ce point d'entree est equivalent a :NSUBR suivi d'un POPJ.

8.3 Les Fonctions standards du compilateur

Toutes les fonctions standards qui vont etre decrites sont de type AUTOLOAD. Elles se trouvent sur un fichier de nom COMPIL.

(COMPILES l) [EXPR a 1 argument]

cre une liste contenant les instructions LAP necessaire a l'execution de la S-expression l. COMPILES ramene cette liste en valeur. Cette foncton est principalement utilise pour tester le fonctionnement du compilateur.

(COMPILE atome sw1 sw2 sw3) [FEXPR]

est la fonction de compilation en mode conversationnel. Cette fonction compile assemble et charge une fonction quelconque. La fonction a compiler doit posseder une definition de type EXPR ou FEXPR. L'indicateur sw1 est egal a T si vous voulez un listing du resultat de la compilation; l'indicateur sw2 sera transmis au LAP et sw3 au chargeur.

(COMPILEFILE filout filin) [SUBR a 2 arguments]

Cette fonction cree un fichier de sortie, de specification filout, contenant la traduction sous forme d'appel de la fonction LAP, de toutes les definitions de fonctions contenues dans le fichier d'entree de specification filin. Le fichier ainsi obtenu pourra etre relu par LISP. Les S-expressions non compilables sont remises (au meme endroit) dans le fichier de sortie.

(COMPILEF file) [FEXPR]

est utilise pour la compilation d'un fichier standard. (COMPILEF file) est la forme abregee de (COMPILEFILE '(DSK (file . LAP)) '(DSK (file . VLI))).

(COMPILEND) [EXPR a 0 argument]

recupere la place occupee par les fonctions du compilateur. COMPILEND remet les indicateurs AUTOLOADs des fonctions standards de et ramene le nombre de doublets liberes en valeur. Cette fonction n'a pas d'effet si vous utilisez le compilateur compile.

8.4 Exemples d'utilisation du compilateur

```
?  (DE FACT (N) (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))
=  FACT
=  ; time = 0 ms ;

?  (TYPEFN 'FACT)
=  EXPR
=  ; time = 0 ms ;

?  (FACT 3)
=  6
=  ; time = 0 ms ;

?  (FACT 6)
=  720
=  ; time = 0 ms ;

?  (COMPILE FACT T T T)

  ; FACT-------------------------------------------------------------

   (DE FACT (N) (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))

   FUNCTION LENGTH = 19
   *LAP LENGTH = 8
  ;
   ( LAP '(

  ;;;;;;
          (ENTRY FACT SUBR 1)
          (CAIN 1 '0)
          (JRST 0 :CRAONE)
   G101
          (PUSH P %T1)       ; (XWD -1 TIMES) ;
          (PUSH P 1)
          (PUSHJ P SUB1)
          (PUSHJ P FACT)
          (JRST 0 :NSUBRP)

  ;---------- * T B L
   *TBL LENGTH = 1;

   %T1        (XWD -1 TIMES)

   ) NIL NIL )

   0             (ENTRY FACT SUBR 1)
   0             (CAIN 1 '0)
   1             (JRST 0 :CRAONE)
   2     G101
   2             (PUSH P %T1)
   3             (PUSH P 1)
```

```
4             (PUSHJ P SUB1)
5             (PUSHJ P FACT)
6             (JRST 0 :NSUBRP)
7    %T1
7             (XWD -1 TIMES)

75362  = 75371                  (VALAP %T1 (7))
75362  = 75364                  (VALAP G101 (2))
75362                           (ENTRY FACT SUBR 1)
75362  306 1  0 0  13660        (CAIN 1 '0)
75363  254 0  0 0  402222       (JRST 0 :CRAONE)
75364  261 17 0 0  75371        (PUSH P %T1)
75365  261 17 0 0  1            (PUSH P 1)
75366  260 17 0 0  407300       (PUSHJ P SUB1)
75367  260 17 0 0  75362        (PUSHJ P FACT)
75370  254 0  0 0  410755       (JRST 0 :NSUBRP)
75371      -1      407335       (XWD -1 TIMES)
75372

(75362 FACT SUBR 1)
                                (END)
=  COMPILE
=  ; time = 540 ms ;

?  (TYPEFN 'FACT)
=  SUBR
=  ; time = 0 ms ;

?  (FACT 3)
=  6
=  ; time = 0 ms ;

?  (FACT 6)
=  720
=  ; time = 0 ms ;

?  (DE ID1 (L) (IF (NULL L) NIL L))
=  ID1
=  ; time = 0 ms ;

?  (COMPILE ID1 T NIL T)

  ; ID1------------------------------------------------------------------------

   (DE ID1 (L) (IF (NULL L) NIL L))

   FUNCTION LENGTH = 12
   *LAP LENGTH = 2
  ;
   ( LAP '(

  ;;;;;;
          (ENTRY ID1 SUBR 1)
          (POPJ P)
```

```
   ) NIL NIL )

   75372                         (ENTRY ID1 SUBR 1)
   75372  263 17 0 0  0          (POPJ P)
   75373

   (75372 ID1 SUBR 1)
                                 (END)
=  COMPILE
=  ; time = 200 ms ;

?  (DE KWOTE (L) (IF (LISTP L) L [QUOTE L]))
=  KWOTE
=  ; time = 0 ms ;

?  (COMPILE KWOTE T NIL T)

  ; KWOTE---------------------------------------------------------

   (DE KWOTE (L) (IF (LISTP L) L [QUOTE L]))

   FUNCTION LENGTH = 15
   *LAP LENGTH = 6
  ;
   ( LAP '(

  ;;;;;;
         (ENTRY KWOTE SUBR 1)
         (CAML 1 :BLIST)
         (POPJ P)
   G101
         (PUSHJ P NCONS)
         (MOVEI 2 'QUOTE)
         (JRST 0 XCONS)

   ) NIL NIL )

   75413  = 75415                (VALAP G101 (2))
   75413                         (ENTRY KWOTE SUBR 1)
   75413  311 1  0 0  1260       (CAML 1 :BLIST)
   75414  263 17 0 0  0          (POPJ P)
   75415  260 17 0 0  406202     (PUSHJ P NCONS)
   75416  201 2  0 0  22         (MOVEI 2 'QUOTE)
   75417  254 0  0 0  406174     (JRST 0 XCONS)
   75420

   (75413 KWOTE SUBR 1)
                                 (END)
=  COMPILE
=  ; time = 280 ms ;

?  (KWOTE 'A)
=  'A
=  ; time = 0 ms ;
```

```
?  (KWOTE ''A)
=  'A
=  ; time = 0 ms ;

?  (DE RECH (L) (IF (LISTP L) (IF (NUMBP (CAR L)) 'QUOTE '**UND)
?  (IF (OR (STRINGP L) (MEMQ L '(NIL T QUOTE LAMBDA SUBR))) NIL
?  (PRINT L "Co to Jest ?" (FOO 'T)) (CASSQ L X))))
=  RECH
=  ; time = 0 ms ;

?  (COMPILE RECH T () T)

  ; RECH-----------------------------------------------------------------------

   (DE RECH (L)
      (IF (LISTP L)
         (IF (NUMBP (CAR L)) 'QUOTE '**UND)
         (IF (OR (STRINGP L) (MEMQ L '(NIL T QUOTE LAMBDA SUBR)))
            NIL
            (PRINT L "Co to Jest ?" (FOO 'T))
            (CASSQ L X))))

   FUNCTION LENGTH = 55
   *LAP LENGTH = 34
  ;
   ( LAP '(

  ;;;;;;
         (ENTRY RECH SUBR)
         (JSP L :SBIND)
         (XWD 0 '(L))
         (GETVAL 1 L)
         (CAMGE 1 :BLIST)
         (JRST 0 G101)
         (CAR 1 1)
         (CAML 1 :BNUMB)
         (CAML 1 :BSTRG)
         (JRST 0 G103)
         (SKIPA 1 %T1)        ; 'QUOTE ;
   G103
         (MOVEI 1 '**UND)
         (POPJ P)
   G101
         (CAML 1 :BSTRG)
         (CAML 1 :BLIST)
         (SKIPA)
         (JRST 0 :FALSE)
         (CAIE 1 'NIL)
         (CAIN 1 'T)
         (JRST 0 :FALSE)
         (CAIE 1 'QUOTE)
         (CAIN 1 'LAMBDA)
         (JRST 0 :FALSE)
         (CAIN 1 'SUBR)
```

```
        (JRST 0 :FALSE)
G105
        (PUSH P %T2)       ; (XWD -1 PRINT) ;
        (PUSH P 1)
        (PUSH P %T3)   .   ; '"Co to Jest ?" ;
        (MOVEI 1 '(FOO 'T))
        (PUSHJ P EVAL)
        (JSP L :NSUBR)
        (GETVAL 1 L)
        (GETVAL 2 X)
        (JRST 0 CASSQ)

;---------- * T B L
*TBL LENGTH = 3;

%T1       'QUOTE
%T2       (XWD -1 PRINT)
%T3       '"Co to Jest ?"

) NIL NIL )

75362  = 75425                (VALAP %T3 (43))
75362  = 75424                (VALAP %T2 (42))
75362  = 75423                (VALAP %T1 (41))
75362  = 75412                (VALAP G105 (30))
75362  = 75376                (VALAP G101 (14))
75362  = 75374                (VALAP G103 (12))
75362                         (ENTRY RECH SUBR)
75362  265 13 0 0   411001    (JSP L :SBIND)
75363      0        26130     (XWD 0 '(L))
75364  554 1  0 0   15070     (HLRZ 1 (+ :MEM 'L))
75365  315 1  0 0   1260      (CAMGE 1 :BLIST)
75366  254 0  0 0   75376     (JRST 0 G101)
75367  554 1  0 1   2154      (HLRZ 1 :MEM 1)
75370  311 1  0 0   1254      (CAML 1 :BNUMB)
75371  311 1  0 0   1257      (CAML 1 :BSTRG)
75372  254 0  0 0   75374     (JRST 0 G103)
75373  334 1  0 0   75423     (SKIPA 1 %T1)
75374  201 1  0 0   10354     (MOVEI 1 '**UND)
75375  263 17 0 0   0         (POPJ P)
75376  311 1  0 0   1257      (CAML 1 :BSTRG)
75377  311 1  0 0   1260      (CAML 1 :BLIST)
75400  334 0  0 0   0         (SKIPA)
75401  254 0  0 0   403237    (JRST 0 :FALSE)
75402  302 1  0 0   0         (CAIE 1 'NIL)
75403  306 1  0 0   6         (CAIN 1 'T)
75404  254 0  0 0   403237    (JRST 0 :FALSE)
75405  302 1  0 0   22        (CAIE 1 'QUOTE)
75406  306 1  0 0   36        (CAIN 1 'LAMBDA)
75407  254 0  0 0   403237    (JRST 0 :FALSE)
75410  306 1  0 0   102       (CAIN 1 'SUBR)
75411  254 0  0 0   403237    (JRST 0 :FALSE)
75412  261 17 0 0   75424     (PUSH P %T2)
75413  261 17 0 0   1         (PUSH P 1)
```

```
75414  261 17 0 0  75425    (PUSH P %T3)
75415  201 1  0 0  26027    (MOVEI 1 '(FOO 'T))
75416  260 17 0 0  403642   (PUSHJ P EVAL)
75417  265 13 0 0  410735   (JSP L :NSUBR)
75420  554 1  0 0  15070    (HLRZ 1 (+ :MEM 'L))
75421  554 2  0 0  15426    (HLRZ 2 (+ :MEM 'X))
75422  254 0  0 0  405746   (JRST 0 CASSQ)
75423      0       22       'QUOTE
75424      -1      403101   (XWD -1 PRINT)
75425      0       22027    '"Co to Jest ?"
75426

(75362 RECH SUBR)
                            (END)
=   COMPILE
=   ; time = 2460 ms ;
```

INDEX

) 25
* 15
** 15
** array error 10
** E.O.F. 25
** E.O.F. during READ. 25
** no room for arrays. 4, 9
** no room for atoms. 4
** no room for code. 4
** no room for lists. 4
** no room for numbers. 4, 13
** no room for strings. 4
** non numeric arg 13
** not enough core. 5
** READ error. 25
** undefined function 12
** undefined symbol 37
*UDS 37
+ 15
- 15
. 13, 25
// 15
/\ 16
1+ 14
1- 14
: 34, 43
:BLIST 32
:BNUMB 32
:BSTRG 32
:CRAFLT 33
:CRANUM 33
:CRAONE 43
:CRASTR 33
:CRAZER 43
:DIFFER 43
:FALSE 43
:GARBCL 33
:MAPC1 43
:MAPCN 43
:MEM 32
:NSUBR 44
:NSUBRP 44
:PLUS 43
:PNAME 33
:QUO 43
:TIMES 43
:TRUTH 43
:TRYATOM 33
:VPOPJ 43
@ 33
APPLY 7
ATAN 16
AUTOLOAD 12
bit 0 du R.G. 2
bit 5 du R.G. 4
bit 7 du R.G. 10
BOUNDP 20
CAR 35, 42
caracteres minuscules 25
caracteres numeriques 25
CDR 36, 42
Chailloux 1
CNTH 7
COMMENT 20
COMPIL 45
compilateur 42
COMPILE 45
COMPILEF 45
COMPILEFILE 45
COMPILEND 45
COMPILES 45
CONFIG.INI 2, 5
CONFIGURATION 2, 5
CONS 36
COPY 20
COS 16
DA 9
DCONS 20
DDT 20
DELETE 20
DELQ 20
DIM 10
DIRECTORY 27
DM 8
END 35
ENTRY 35
EOF 25
ETV 27, 30
EVAL 7
EXP 17, 34
fichier initial 2, 5
FILLARRAY 11
FIX 14
FLOAT 14
FORTRAN 13
FREVERSE 21
FUNCTION 21
Garbage-Collecting 4
GET 23
GETL 23
GETVAL 42
Goossens 1
Greussay 1, 7
Harvey 1
HIGHSEG 2
IFN 21

INUMBP 21
JNLIST 36
JPLIST 36
LAP 31, 37
LAPEND 37
LAPF 37
LAST 19
LIBRARY 12, 27
LIST 25
LISTARRAY 11
LODLAP 37
LOG 17
LOG10 17
MACLAP 35
MACRO 7
MAPARRAY 9, 10
MAPARRAYQ 11
MCONS 21
MODULO 16
NCONS 21
NIL 32
OBLIST 19, 25
OPCD 34
P-liste 23
P-name 25
PATHLIBRARY 27
pile utilisateur 23
POP 24
ppn 26, 27
PRETTY 29
PRETTY-PRINT 29
PRETTYEND 30
PRETTYF 30
PRETTYFILE 29
PRETTYP 29
PRETTYSIZE 30
PRIN1 29
PRINT 26, 29
PUSH 23
PUT 23
PUTVAL 42
QUOTE 26, 34
RANDOM 18
RDCORE 28
READ 25
registre 31
REMPROP 23
repertoire 26, 27
REVERSE 19
SAMEPN 22
SETA 10
SETNIL 42
SETQA 10
SIN 16
SQRT 16
SUBST 20
tableau 9
tail-recursives 7
TOPLEVEL 2
TYPEFN 22
TYPEP 22
TYPNUMB 23
UNCONS 36
UNDEF 19
VALAP 34
Wertz 1
WRCORE 28
XCONS 22
XWD 34
[25
\ 26
] 25
^ G.C. ^ type .CONT please. 4
^C 4